1U－02

実行不要命令の削除によるコード最適化

中嶋 廣二　小谷 謙介　佐山 旬子　田中 旭
松下電器産業㈱ 半導体開発本部 マイコン開発センター
第四開発グループ 第一開発チーム

1. はじめに

近年、組込みマイコン分野におけるソフトウェア開発においては、開発効率や保守性からC言語などの高級言語による開発が主流になってきている。特に組込みマイコン分野では、ハードウェアの制約上、コードサイズや実行時間に制限がある場合が多く、コンパイラによるコード最適化が重要となってくる。

コード最適化の一手法として実行不要命令の削除がある。これは、コンパイラが生成した目的プログラムにおいて、削除してもソースプログラムの実行結果が変化しない命令を削除する最適化である。この最適化の従来手法としては、格納されている値と同一の値を転送する命令を削除する手法や、参照されない資源定義命令を削除する手法などがある。これらの手法では、削除しても他の命令に全く影響がない命令のみ削除している。しかし実際には削除した場合に一部の命令の結果は変化する可能性があるが、プログラムの結果は変化させない命令も存在する。このような命令はプログラムの結果は変化していないという意味で広義の実行不要命令であると言える。本稿ではこの広義の実行不要命令を削除する手法について述べる。

2. 実行不要命令の削除

ここでは、広義の実行不要命令を削除する手法について述べる。まず、コンパイラが生成した目的プログラムの各命令を以下の2種類に分類する。

・結果命令：ソースプログラムの結果を表す命令
　　… メモリストア、分岐実行等
・中間命令：ソースプログラムの結果を求める過程の命令 … メモリロード、演算命令等

中間命令と結果命令の例を図1に示す。

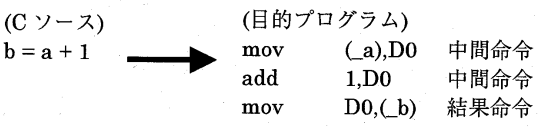

図1 中間命令と結果命令

ある命令iが実行不要命令かどうかを解析する手法は以下の3ステップからなる。

ステップ1：命令iを実行しないことによって結果が変化する可能性がある命令をすべて抽出
ステップ2：ステップ1で抽出した命令について実行結果が変化するかどうかを解析
ステップ3：ソースプログラムの結果が変化しないかどうかを解析

ステップ1では、命令iで定義する資源を参照する全命令を抽出する。抽出した命令が中間命令のときは、その命令の定義資源を参照する全ての命令を再帰的に抽出する。また資源の定義参照関係は有向グラフで表す。有向辺は資源を定義する命令から参照する命令に向けて引く。図2にグラフの例を示す。

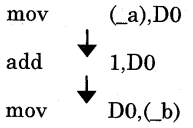

図2 定義参照関係を表す有向グラフ

ステップ2では、ステップ1で抽出した命令について命令iを削除することによる実行結果の変化の有無を解析する。各命令の結果はレジスタやメモリの値1ビット毎に求める。これにより定義した資源の一部のビットしか参照されない命令について、参照される部分のみの変化の有無を判定することが可能となる。また、コンパイル時にはレジスタの値が完全にわからないので、各ビットの値表現には、0, 1の値だけでなく、値が不明であることや、命令の削除前後で値が等しいかどうかなどを表す状態表現を用いる。状態表現の一覧を図3に示す。

各命令の実行結果変化の解析においては、命令iを実行した場合と削除した場合の結果をそれぞれ求める。さらに命令の種類に応じて結果の変化の有無

A code optimization to delete unnecessary instructions
Koji Nakajima, Kensuke Odani, Junko Sayama, Akira Tanaka
Development Team 1 Development Group 4
Microprocessor Development Center
Corporate Semiconductor Development Division
Matsushita Electric Industrial Co., LTD.

状態表現	意味
0E	ビットの値は0、値は等しい
0N	ビットの値は0、値は異なる
1E	ビットの値は1、値は等しい
1N	ビットの値は1、値は異なる
UE	ビットの値は不明、値は等しい
UN	ビットの値は不明、値は異なる

図3 状態表現の一覧

を解析する(例:定数0との論理積はレジスタの値に依らず0)。図4に実行結果変化の解析例を示す。

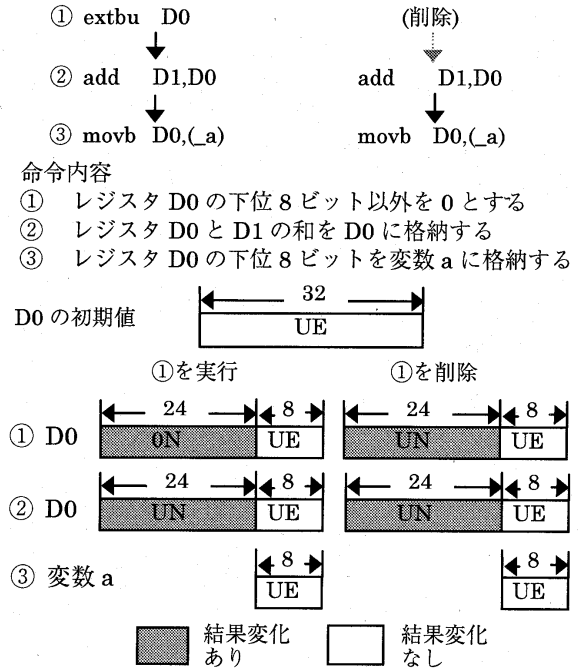

図4 実行結果変化の解析

①の実行前のD0はコンパイル時に値が不明であり、また値が①の実行には依存しないため、全ビットUEとする。D0の上位24ビットについては①を実行した場合は0、削除した場合は不明であり値は異なる。したがって上位24ビットについては①を実行した場合は0N、削除した場合はUNとなる。下位8ビットは等しいので共にUEとなる。②では、加算命令は最下位ビットから順に結果を計算する。D1の値はコンパイル時に不明、かつ値が①の実行に依存しないので全ビットUEである。下位8ビットの各ビットはUE+UEである。この場合、加算する両値は等しいので結果も等しく共にUEである。上位24ビットは、①を実行する場合は0N+UE、①を削除する場合はUN+UEとなる。結果は不明だが、加算する値が異なるため結果は異なり共にUNとなる。③は②の結果の下位8ビットを変数aに格納する。なお、ここではD0,D1の初期値を全ビットUEとしているが、定義先を解析することにより値を推測し、結果をより詳細に解析することも可能である。

ステップ3では、命令iを削除することにより、ソースプログラムの結果が変化するかどうかを調べる。ソースプログラムの結果が変化するのは、結果命令の結果が変化するときのみである。したがって、ステップ1で抽出した命令のうち、全ての結果命令の結果が変化していないとき、命令iは実行不要命令であると言える。図4では、中間命令である②の結果のうち下位8ビット以外は変化しているが、結果命令である③は②の結果のうち下位8ビットしか参照しておらず結果が変化していない。したがって①は実行不要命令であると言える。

3. コードサイズ評価

ここでは、2章で述べた最適化手法を、当社32ビットマイコンAM30用コンパイラcc103において不要な型変換命令の削除に適用し、評価を行った結果を述べる。型変換命令とは、8または16ビットのデータを32ビットレジスタで扱うために行う命令である(図4の①など)。

今回は家電用のサンプルプログラムやベンチマークプログラム等、13種類のプログラムについてコードサイズの削減率を測定した。結果を図5に示す。

削減率	ソース数
0～1%	10
1～2%	1
2%以上	2

図5 コードサイズ削減率

最大7.8%、平均1.1%のコードサイズが削減された。int演算のみのプログラムでは型変換命令が生成されず、効果がほとんど見られなかったが、char演算やshort演算が連続するプログラムでは大きな効果が得られることがわかった。

4. まとめ

今回は、広義の実行不要命令の削除最適化法と不要型変換命令削除への適用例について述べた。今後は、対象を型変換以外の命令へ広げ、さらなるコード改善を行う予定である。